



C interfaces to GALAHAD LSRT

Jari Fowkes and Nick Gould
STFC Rutherford Appleton Laboratory
Thu Jun 22 2023

| | |
|------------------------------------|-----------|
| 1 GALAHAD C package lsrt | 1 |
| 1.1 Introduction | 1 |
| 1.1.1 Purpose | 1 |
| 1.1.2 Authors | 1 |
| 1.1.3 Originally released | 1 |
| 1.1.4 Terminology | 1 |
| 1.1.5 Method | 2 |
| 1.1.6 Reference | 2 |
| 1.1.7 Call order | 3 |
| 2 File Index | 5 |
| 2.1 File List | 5 |
| 3 File Documentation | 7 |
| 3.1 galahad_lsrt.h File Reference | 7 |
| 3.1.1 Data Structure Documentation | 7 |
| 3.1.1.1 struct lsrt_control_type | 7 |
| 3.1.1.2 struct lsrt_inform_type | 8 |
| 3.1.2 Function Documentation | 9 |
| 3.1.2.1 lsrt_initialize() | 9 |
| 3.1.2.2 lsrt_read_specfile() | 9 |
| 3.1.2.3 lsrt_import_control() | 10 |
| 3.1.2.4 lsrt_solve_problem() | 10 |
| 3.1.2.5 lsrt_information() | 12 |
| 3.1.2.6 lsrt_terminate() | 12 |
| 4 Example Documentation | 13 |
| 4.1 lsrtt.c | 13 |

Chapter 1

GALAHAD C package lsrt

1.1 Introduction

1.1.1 Purpose

Given a real m by n matrix A , a real m vector b and scalars $\sigma > 0$ and $p \geq 2$, this package finds an **approximate minimizer of the regularized linear-least-squares objective function** $\frac{1}{2}\|Ax - b\|_2^2 + \frac{1}{p}\sigma\|x\|_2^p$. This problem commonly occurs as a subproblem in nonlinear optimization calculations involving cubic regularization, and may be used to regularize the solution of under-determined or ill-conditioned linear least-squares problems. The method may be suitable for large m and/or n as no factorization involving A is required. Reverse communication is used to obtain matrix-vector products of the form $u + Av$ and $v + A^T u$.

1.1.2 Authors

N. I. M. Gould, STFC-Rutherford Appleton Laboratory, England.

C interface, additionally J. Fowkes, STFC-Rutherford Appleton Laboratory.

Julia interface, additionally A. Montoison and D. Orban, Polytechnique Montréal.

1.1.3 Originally released

November 2007, C interface December 2021.

1.1.4 Terminology

The required solution x necessarily satisfies the optimality condition $A^T(Ax - b) + \lambda x = 0$, where the multiplier $\lambda = \sigma\|x\|_2^{p-2}$.

1.1.5 Method

The method is iterative. Starting with the vector $u_1 = b$, a bi-diagonalisation process is used to generate the vectors v_k and u_{k+1} so that the n by k matrix $V_k = (v_1 \dots v_k)$ and the m by $(k+1)$ matrix $U_k = (u_1 \dots u_{k+1})$ together satisfy

$$AV_k = U_{k+1}B_k \text{ and } b = \|b\|_2 U_{k+1}e_1$$

where B_k is $(k+1)$ by k and lower bi-diagonal, U_k and V_k have orthonormal columns and e_1 is the first unit vector. The solution sought is of the form $x_k = V_k y_k$, where y_k solves the bi-diagonal regularized least-squares problem

$$(1) \quad \min \frac{1}{2} \|B_k y - \|b\|_2 e_1\|_2^2 + \frac{1}{p} \sigma \|y\|_2^p.$$

To minimize (1), the optimality conditions

$$(B_k^T (B_k y(\lambda) - \|b\|_2 e_1) + \lambda y(\lambda) = 0,$$

where $\lambda = \sigma \|y(\lambda)\|_2^{p-2}$, are used as the basis of an iteration. The vector $y(\lambda)$ is equivalently the solution to the regularized least-squares problem

$$(2) \quad \min \left\| \begin{pmatrix} B_k \\ \lambda^{\frac{1}{p}} I \end{pmatrix} y - \|b\|_2 e_1 \right\|_2.$$

Thus, given an estimate $\lambda \geq 0$, (2) may be efficiently solved to give $y(\lambda)$. It is then simply a matter of adjusting λ (for example by a Newton-like process) to solve the scalar nonlinear equation

$$(3) \quad \theta(\lambda) \equiv \|y(\lambda)\|_2^{p-2} - \frac{\lambda}{\sigma} = 0.$$

In practice (3) is reformulated, and a more rapidly converging iteration is used. Having found y_k , a second pass in which $x_k = V_k y_k$ is regenerated is needed—this need only be done once x_k has implicitly deemed to be sufficiently close to optimality. As this second pass is an additional expense, a record is kept of the optimal objective function values for each value of k , and the second pass is only performed so far as to ensure a given fraction of the final optimal objective value. Large savings may be made in the second pass by choosing the required fraction to be significantly smaller than one.

Special code is used in the special case $p = 2$, as in this case a single pass suffices.

1.1.6 Reference

A complete description of the un- and quadratically-regularized cases is given by

C. C. Paige and M. A. Saunders, LSQR: an algorithm for sparse linear equations and sparse least squares. ACM Transactions on Mathematical Software, 8(1):43–71, 1982

and

C. C. Paige and M. A. Saunders, ALGORITHM 583: LSQR: an algorithm for sparse linear equations and sparse least squares. ACM Transactions on Mathematical Software, 8(2):195–209, 1982.

Additional details on the Newton-like process needed to determine λ and other details are described in

C. Cartis, N. I. M. Gould and Ph. L. Toint, Trust-region and other regularisation of linear least-squares problems. BIT 49(1):21-53 (2009).

1.1.7 Call order

To solve a given problem, functions from the lsrt package must be called in the following order:

- [lsrt_initialize](#) - provide default control parameters and set up initial data structures
- [lsrt_read_specfile](#) (optional) - override control values by reading replacement values from a file
- [lsrt_import_control](#) - import control parameters prior to solution
- [lsrt_solve_problem](#) - solve the problem by reverse communication, a sequence of calls are made under control of a status parameter, each exit either asks the user to provide additional information and to re-enter, or reports that either the solution has been found or that an error has occurred
- [lsrt_information](#) (optional) - recover information about the solution and solution process
- [lsrt_terminate](#) - deallocate data structures

See Section [4.1](#) for an example of use.

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

| | |
|--|---|
| galahad_lsrt.h | 7 |
|--|---|

Chapter 3

File Documentation

3.1 galahad_lsrt.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
```

Data Structures

- struct [lsrt_control_type](#)
- struct [lsrt_inform_type](#)

Functions

- void [lsrt_initialize](#) (void **data, struct [lsrt_control_type](#) *control, int *status)
- void [lsrt_read_specfile](#) (struct [lsrt_control_type](#) *control, const char specfile[])
- void [lsrt_import_control](#) (struct [lsrt_control_type](#) *control, void **data, int *status)
- void [lsrt_solve_problem](#) (void **data, int *status, int m, int n, const real_wp_ power, const real_wp_ weight, real_wp_ x[], real_wp_ u[], real_wp_ v[])
- void [lsrt_information](#) (void **data, struct [lsrt_inform_type](#) *inform, int *status)
- void [lsrt_terminate](#) (void **data, struct [lsrt_control_type](#) *control, struct [lsrt_inform_type](#) *inform)

3.1.1 Data Structure Documentation

3.1.1.1 struct [lsrt_control_type](#)

control derived type as a C struct

Examples

[lsrtt.c](#).

Data Fields

| | | |
|----------|------------------------|---|
| bool | f_indexing | use C or Fortran sparse matrix indexing |
| int | error | error and warning diagnostics occur on stream error |
| int | out | general output occurs on stream out |
| int | print_level | the level of output required is specified by print_level |
| int | start_print | any printing will start on this iteration |
| int | stop_print | any printing will stop on this iteration |
| int | print_gap | the number of iterations between printing |
| int | itmin | the minimum number of iterations allowed (-ve = no bound) |
| int | itmax | the maximum number of iterations allowed (-ve = no bound) |
| int | bitmax | the maximum number of Newton inner iterations per outer iteration allowed (-ve = no bound) |
| int | extra_vectors | the number of extra work vectors of length n used |
| int | stopping_rule | the stopping rule used: 0=1.0, 1=norm step, 2=norm step/sigma (NOT USED) |
| int | freq | frequency for solving the reduced tri-diagonal problem (NOT USED) |
| real_wp_ | stop_relative | the iteration stops successfully when $ A^{Tr} $ is less than $\max(\text{stop_relative} * A^{Tr} \text{ initial} , \text{stop_absolute})$ |
| real_wp_ | stop_absolute | see stop_relative |
| real_wp_ | fraction_opt | an estimate of the solution that gives at least .fraction_opt times the optimal objective value will be found |
| real_wp_ | time_limit | the maximum elapsed time allowed (-ve means infinite) |
| bool | space_critical | if .space_critical true, every effort will be made to use as little space as possible. This may result in longer computation time |
| bool | deallocate_error_fatal | if .deallocate_error_fatal is true, any array/pointer deallocation error will terminate execution. Otherwise, computation will continue |
| char | prefix[31] | all output lines will be prefixed by .prefix(2:LEN(TRIM(.prefix))-1) where .prefix contains the required string enclosed in quotes, e.g. "string" or 'string' |

3.1.1.2 struct lsrt_inform_type

inform derived type as a C struct

Examples

[lsrtt.c](#).

Data Fields

| | | |
|------|---------------|---|
| int | status | return status. See lsrt_solve_problem for details |
| int | alloc_status | the status of the last attempted allocation/deallocation |
| char | bad_alloc[81] | the name of the array for which an allocation/deallocation error occurred |
| int | iter | the total number of iterations required |
| int | iter_pass2 | the total number of pass-2 iterations required |
| int | biters | the total number of inner iterations performed |
| int | biter_min | the smallest number of inner iterations performed during an outer iteration |
| int | biter_max | the largest number of inner iterations performed during an outer iteration |

Data Fields

| | | |
|----------|------------|--|
| real_wp_ | obj | the value of the objective function |
| real_wp_ | multiplier | the multiplier, $\lambda = \sigma \ x\ ^{(p-2)}$ |
| real_wp_ | x_norm | the Euclidean norm of x |
| real_wp_ | r_norm | the Euclidean norm of $Ax - b$ |
| real_wp_ | Atr_norm | the Euclidean norm of $A^T(Ax - b) + \lambda x$ |
| real_wp_ | biter_mean | the average number of inner iterations performed during an outer iteration |

3.1.2 Function Documentation

3.1.2.1 lsrt_initialize()

```
void lsrt_initialize (
    void ** data,
    struct lsrt_control_type * control,
    int * status )
```

Set default control values and initialize private data

Parameters

| | | |
|---------|----------------|--|
| in, out | <i>data</i> | holds private internal data |
| out | <i>control</i> | is a struct containing control information (see lsrt_control_type) |
| out | <i>status</i> | is a scalar variable of type int, that gives the exit status from the package. Possible values are (currently): <ul style="list-style-type: none"> • 0. The import was succesful. |

Examples

[lsrtt.c](#).

3.1.2.2 lsrt_read_specfile()

```
void lsrt_read_specfile (
    struct lsrt_control_type * control,
    const char specfile[] )
```

Read the content of a specification file, and assign values associated with given keywords to the corresponding control parameters. By default, the spcification file will be named RUNLSRT.SPC and lie in the current directory. Refer to Table 2.1 in the fortran documentation provided in \$GALAHAD/doc/lsrt.pdf for a list of keywords that may be set.

Parameters

| | | |
|----------------|-----------------|---|
| <i>in, out</i> | <i>control</i> | is a struct containing control information (see lsrt_control_type) |
| <i>in</i> | <i>specfile</i> | is a character string containing the name of the specification file |

3.1.2.3 lsrt_import_control()

```
void lsrt_import_control (
    struct lsrt\_control\_type * control,
    void ** data,
    int * status )
```

Import control parameters prior to solution.

Parameters

| | | |
|----------------|----------------|---|
| <i>in</i> | <i>control</i> | is a struct whose members provide control paramters for the remaining prcedures (see lsrt_control_type) |
| <i>in, out</i> | <i>data</i> | holds private internal data |
| <i>in, out</i> | <i>status</i> | is a scalar variable of type int, that gives the exit status from the package. Possible values are (currently): <ul style="list-style-type: none"> 1. The import was succesful, and the package is ready for the solve phase |

Examples

[lsrtt.c](#).

3.1.2.4 lsrt_solve_problem()

```
void lsrt_solve_problem (
    void ** data,
    int * status,
    int m,
    int n,
    const real_wp_ power,
    const real_wp_ weight,
    real_wp_ x[],
    real_wp_ u[],
    real_wp_ v[] )
```

Solve the regularized least-squares problem using reverse communication.

Parameters

| | | |
|----------------|-------------|-----------------------------|
| <i>in, out</i> | <i>data</i> | holds private internal data |
|----------------|-------------|-----------------------------|

Parameters

| | | |
|----------------------|---------------------|--|
| <code>in, out</code> | <code>status</code> | <p>is a scalar variable of type int, that gives the entry and exit status from the package. This must be set to</p> <ul style="list-style-type: none"> 1. on initial entry. Set u (below) to b for this entry. <p>Possible exit values are:</p> <ul style="list-style-type: none"> 0. the solution has been found 2. The user must perform the operation $u := u + Av,$ <p>and recall the function. The vectors u and v are available in the arrays u and v (below) respectively, and the result u must overwrite the content of u. No argument except u should be altered before recalling the function</p> 3. The user must perform the operation $v := v + A^T u,$ <p>and recall the function. The vectors u and v are available in the arrays u and v (below) respectively, and the result v must overwrite the content of v. No argument except v should be altered before recalling the function</p> 4. The user must reset u (below) to b and recall the function. No argument except u should be altered before recalling the function -1. an array allocation has failed -2. an array deallocation has failed -3. one or more of n, m, power or weight violates allowed bounds -18. the iteration limit has been exceeded -25. status is negative on entry |
| <code>in</code> | <code>m</code> | is a scalar variable of type int, that holds the number of equations (i.e., rows of A), $m > 0$ |
| <code>in</code> | <code>n</code> | is a scalar variable of type int, that holds the number of variables (i.e., columns of A), $n > 0$ |
| <code>in</code> | <code>power</code> | is a scalar of type double, that holds the regularization power, $p \geq 2$ |
| <code>in</code> | <code>weight</code> | is a scalar of type double, that holds the regularization weight, $\sigma > 0$ |
| <code>in, out</code> | <code>x</code> | is a one-dimensional array of size n and type double, that holds the solution x . The j -th component of x , $j = 0, \dots, n-1$, contains x_j . |
| <code>in, out</code> | <code>u</code> | is a one-dimensional array of size m and type double, that should be used and reset appropriately when status = 1 to 5 as directed by status. |
| <code>in, out</code> | <code>v</code> | is a one-dimensional array of size n and type double, that should be used and reset appropriately when status = 1 to 5 as directed by status. |

Examples

[lsrtt.c.](#)

3.1.2.5 lsrt_information()

```
void lsrt_information (
    void ** data,
    struct lsrt_inform_type * inform,
    int * status )
```

Provides output information

Parameters

| | | |
|---------|---------------|---|
| in, out | <i>data</i> | holds private internal data |
| out | <i>inform</i> | is a struct containing output information (see lsrt_inform_type) |
| out | <i>status</i> | is a scalar variable of type int, that gives the exit status from the package. Possible values are (currently): <ul style="list-style-type: none"> • 0. The values were recorded succesfully |

Examples

[lsrtt.c](#).

3.1.2.6 lsrt_terminate()

```
void lsrt_terminate (
    void ** data,
    struct lsrt_control_type * control,
    struct lsrt_inform_type * inform )
```

Deallocate all internal private storage

Parameters

| | | |
|---------|----------------|---|
| in, out | <i>data</i> | holds private internal data |
| out | <i>control</i> | is a struct containing control information (see lsrt_control_type) |
| out | <i>inform</i> | is a struct containing output information (see lsrt_inform_type) |

Examples

[lsrtt.c](#).

Chapter 4

Example Documentation

4.1 lsrtt.c

This is an example of how to use the package to solve a regularized quadratic problem. The use of default and non-default scaling matrices, and restarts with a larger regularization weight are illustrated.

```
/* lsrtt.c */
/* Full test for the LSRT C interface */
#include <stdio.h>
#include <math.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
#include "galahad_lsrt.h"
int main(void) {
    // Derived types
    void *data;
    struct lsrt_control_type control;
    struct lsrt_inform_type inform;
    // Set problem data
    int n = 50; // dimensions
    int m = 2 * n;
    int status;
    real_wp_ power = 3.0;
    real_wp_ weight = 1.0;
    real_wp_ x[n];
    real_wp_ u[m];
    real_wp_ v[n];
    // Initialize lsrt
    lsrt_initialize( &data, &control, &status );
    status = 1;
    control.print_level = 0;
    lsrt_import_control( &control, &data, &status );
    for( int i = 0; i < m; i++) u[i] = 1.0; // b = 1
    // iteration loop to find the minimizer with  $A^T = (I:\text{diag}(1:n))$ 
    while(true){ // reverse-communication loop
        lsrt_solve_problem( &data, &status, m, n, power, weight, x, u, v );
        if ( status == 0 ) { // successful termination
            break;
        } else if ( status < 0 ) { // error exit
            break;
        } else if ( status == 2 ) { // form  $u \leftarrow u + A * v$ 
            for( int i = 0; i < n; i++) {
                u[i] = u[i] + v[i];
                u[n+i] = u[n+i] + (i+1)*v[i];
            }
        } else if ( status == 3 ) { // form  $v \leftarrow v + A^T * u$ 
            for( int i = 0; i < n; i++) v[i] = v[i] + u[i] + (i+1) * u[n+i];
        } else if ( status == 4 ) { // restart
            for( int i = 0; i < m; i++) u[i] = 1.0;
        } else {
            printf(" the value %i of status should not occur\n",
                status);
            break;
        }
    }
    lsrt_information( &data, &inform, &status );
    printf("lsrt_solve_problem exit status = %i,"
        " f = %.2f\n", inform.status, inform.obj );
    // Delete internal workspace
    lsrt_terminate( &data, &control, &inform );
}
```

